PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 15/16	A1	(11) International Publication Number: WO 98/09225
		(43) International Publication Date: 5 March 1998 (05.03.98)

(21) International Application Number: PCT/US97/13182

(22) International Filing Date: 28 July 1997 (28.07.97)

(30) Priority Data:
08/705,285 29 August 1996 (29.08.96) US

(71) Applicant: NEMATRON CORPORATION [US/US]; 5840 Interface Drive, Ann Arbor, MI 48103 (US).

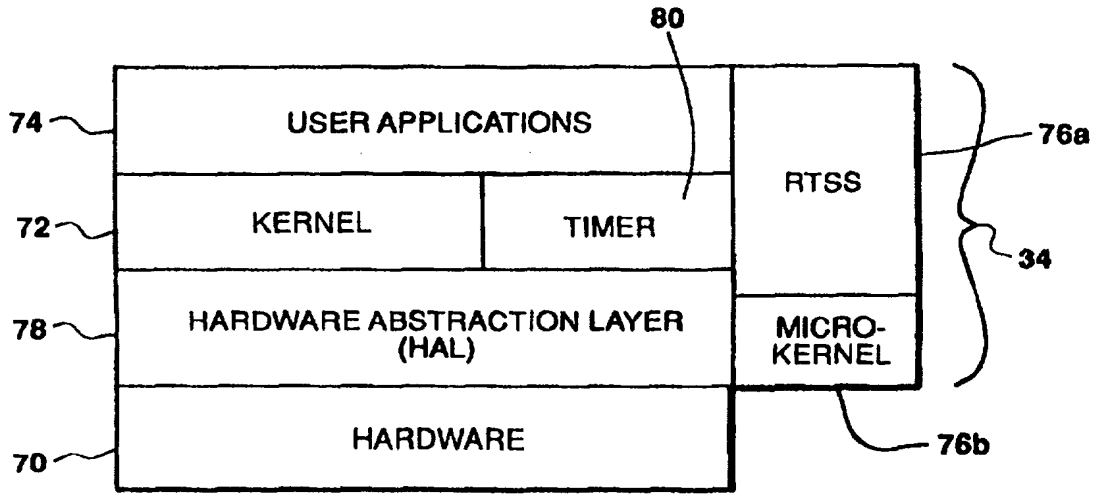
(72) Inventors: LOGAN, Frank, G., III; 5152 Pheasant Trail, Ann Arbor, MI 48105 (US). ACHESINSKI, Jeffrey, M.; 3709 Sterncroft Court, Virginia Beach, VA 23456 (US). DAVIS, Teddy, M.; 1845 Arrowwood Street, Norfold, VA 23518 (US).

(74) Agents: MILTON, Harold, W., Jr. et al.; Howard & Howard Attorneys, P.C., Suite 101, 1400 North Woodward Avenue, Bloomfield Hills, MI 48304-2856 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: REAL TIME SOFTWARE SYSTEM



(57) Abstract

A computer system (10) and a method of operation thereof includes a memory (34) having a standard operating system stored in a first portion (72) and a real time software system stored in a second portion (76a, 76b), and a CPU (20) connected to the memory. The standard operating system controls the CPU (20) to generate standard time slices (100) each having a plurality of instruction cycles (82) and the real time sofware system controls the CPU to divide the standard time slices into real time software system time slices (108) and standard operating system time slices (110). Real time program instructions are executed during the real time software system time slices (108) and other program instructions are executed during the standard operating system time slices (110). The real time software system can select a duty cycle for the ratio of the number of the time slices (108, 110) in the stardard time slice (100). A plurality of input/ouput interrupts can be serviced on a real time basis during the real time software system time slices (108).

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

REAL TIME SOFTWARE SYSTEM

BACKGROUND OF THE INVENTION

The present invention relates generally to a method and apparatus for operating computer systems and, in particular, to a method and apparatus for operating a computer system on a real time basis.

During the operation of computer systems, various internal events and input/output (I/O) events occur that require service by the system processor. In a typical microprocessor based computer, devices such as timers and memory controllers generate events which require service. Also, an I/O bus can be connected to peripheral devices requiring service such as a network communication adapter card, a printer interface, a keyboard interface, a mouse interface, a floppy drive, a hard drive, a CD-ROM drive, a tape drive, etc.

Two common techniques for servicing events are polling and interrupt. A processor utilizing the polling technique regularly reads the status of devices that may need service. This checking is done during the normal execution of the operating system program and any applications program. Therefore, polling may not be sufficiently responsive to events which occur asynchronously. Devices which generate a relatively large quantity of data may exceed the available buffer memory between the regular polling times in successive processor polling cycles. Also, polling can result in a large number of processor cycles during which no service operations are necessary.

The interrupt technique is utilized to solve some of the real time disadvantages of the polling technique. When a device requires service, it generates an interrupt signal that causes the processor to execute an interrupt service routine. Thus, the processor stops execution of program instructions, stores the current status information for the program, executes the interrupt service routine and restores the current status information. However, as the number of interrupts in any time period increases, the processor becomes less efficient at running the

current program due to the extra time required to store and reload the current status information at each interrupt.

Device service is especially critical in the control of manufacturing process operations. Much process equipment requires data exchange on a real time basis in order to precisely control variables such as drive motor speed and tool movement paths. Various solutions to the problem of precise process control have been proposed. For example, the U.S. Patent No. 4,228,495 discloses a typical multiprocessor solution for a numerical control system. The system includes a main microprocessor, a programmable interface with an I/O microprocessor and a front panel microprocessor. The main microprocessor performs the functions of interpolation and outputing of motion command signals, the I/O microprocessor operates as a programmable controller to control the process equipment and the front panel microprocessor services the I/O devices associated with the front control panel. The main microprocessor operates under the direction of a scheduler routine which is clocked at 1.6 msec intervals and has a 25.6 msec cycle. The scheduler routine allocates 1.6 msec slices to various processes to be performed including a timed interrupt process that outputs motion commands to the servomechanism interface and exchanges data with the programmable interface. The timed interrupt process is always executed to completion during a scheduler cycle such that the number of 1.6 msec intervals required by it will vary from cycle to cycle.

The U.S. Patent No. 5,163,146 discloses a computer which has an operating system program that includes an interrupt handler module. The program changes the speed of the computer clock to maintain the computer in synchronism with an I/O adapter or to speed up the operation when slower speed circuits are not being utilized. Circuits and programs which require like cycle times are associated with a common interrupt level and the operating program changes the clock speed in response to the current interrupt level.

The U.S. Patent No. 5,437,039 discloses a system management interrupt handler comprising a plurality of service tasks which are executed interleavingly with normal execution. A prior art non-maskable transparent system management

interrupt (SMI) is described in connection with Fig. 1 wherein a SMI places the computer system into an execution mode that is transparent to the operating system. A dedicated memory area not mapped in the computer's memory address space is swapped in to store the system state before the interrupt handler is given control. Interrupt latency in this prior art system is reduced by providing a reserved trigger timer (Fig. 6). If the SMI was not caused by the timer, service tasks associated with the SMI are queued and the computer returns to the normal execution. If the SMI was caused by the timer, the next queued service task is executed and the computer returns to the normal execution.

SUMMARY OF THE INVENTION

The present invention concerns a method and an apparatus for operating a computer system, the computer system having a memory for storing an operating system and a CPU for executing program instructions under control of the operating system. The method according to the present invention includes the steps of: storing a first operating system in a first portion of the memory; operating the CPU under the control of the first operating system to generate a plurality of standard time slices; storing a second operating system in a second portion of the memory; operating the CPU under the control of the second operating system concurrently with the first operating system to generate at least one first time slice and at least one second time slice during each of the standard time slices; executing at least one program instruction in accordance with the first operating system during the first time slice; and executing at least one other program instruction in accordance with the second operating system during the second time slice.

The computer system according to the present invention includes a CPU connected to a memory for executing program instructions according to an operating system and comprises: the memory having a first portion in which a first operating system is stored and a second portion in which a second operating system is stored; and the CPU operating under control of the first operating system to

sequentially generate standard time slices and concurrently operating under control of said second operating system to divide the standard time slices into at least one first time slice and at least one second time slice, the CPU executing first program instructions during the first time slice and executing second program instructions during the second time slice.

The present invention overcomes the disadvantages of prior art process controllers by utilizing a standard personal computer running a commercially available operating system to control a manufacturing process requiring "hard" real time interrupts.

A further advantage of the present invention is that the portion of the standard time slice devoted to "hard" real time operation can be selectively varied to maximize the performance of the control system.

BRIEF DESCRIPTION OF THE DRAWINGS

The above, as well as other advantages of the present invention, will become readily apparent to those skilled in the art from the following detailed description of a preferred embodiment when considered in the light of the accompanying drawings in which:

Fig. 1 is a schematic block diagram of a typical microprocessor based computer system;

Fig. 2 is schematic memory map diagram of a prior art generic operating system in the random access memory shown in the Fig. 1;

Fig. 3 is schematic memory map diagram of a prior art Windows NT operating system in the random access memory shown in the Fig. 1;

Fig. 4 is a timing diagram of a typical instruction cycle for the computer system shown in the Fig. 1;

Fig. 5 is a flow diagram of the operating system instruction cycle shown in the Fig. 4 and an interrupt service routine for the computer system shown in the Fig. 1;

Fig. 6 is a timing diagram of the time slices for the operating systems shown in the Fig. 2 and the Fig. 3;

Fig. 7 is a plot of time slice length versus overhead for the computer system shown in the Fig. 1;

Fig. 8 is schematic memory map diagram similar to the Fig. 3 configured to include a real time software system in accordance with the present invention;

Fig. 9 is a timing diagram of the operating system time slices generated by the real time software system shown in the Fig. 8; and

Fig. 10 is a timing diagram similar to the Fig. 9 with the operating system time slices modified in accordance with the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

There is shown in the Fig. 1 a schematic block diagram of a typical microprocessor based computer system **10**. The system **10** includes an expanded central processing unit **12**, an input/output unit **14** and a graphics interface unit **16** each connected to a main bus **18**. The expanded central processing unit **12** has a central processing unit (CPU) **20** connected to a CPU bus **22**. The expanded central processing unit **12** also includes a bus control **24**, a cache control **26** and a RAM control **28** each connected to the bus **22**. The bus **22** is connected for communication to the main bus **18** through a plurality of line buffers **30**. A cache memory **32**, external to the expanded central processing unit **12**, is connected to and controlled by the cache control **26**. Similarly, a random access memory (RAM) **34** is located external to the expanded central processing unit **12** and is connected to and controlled by the RAM control **28**.

The input/output unit **14** includes parallel I/O ports **36** and serial I/O ports **38** connected to an I/O bus **40** which in turn is connected to the main bus **18**. The input/output unit **14** further includes a programmable interrupt control **42**, a programmable timer **44**, a direct memory access control **46**, a real time clock unit **48** and a hard drive control **50** all connected to the input/output unit bus **40**.

External to the input/output unit **14** and connected to the main bus **18** is a hard drive **52** which is controlled by the hard drive control **50**. The hard drive **52** and the associated control **50** represent any type of mass storage device suitable for use with a microprocessor based computer.

The graphics interface unit **16** includes a video graphics control **54** and a video memory control **56** connected to a graphics bus **58** which in turn is connected to the main bus **18**. External to the graphics interface unit **16** is a video memory **60** which is connected to the video memory control **56** and to the main bus **18**. The video memory **60** can be accessed by the video memory control **56** and through the main bus **18**. A monitor interface **62** is connected between the video memory **60** and a video monitor **64**. Also connected to the main bus **18** are one or more device controls **66** which in turn are connected to associated devices **68**. The controls **66** and the devices **68** can include a network communication adapter card, a printer interface, a keyboard interface, a mouse interface, a floppy drive, a CD-ROM drive, a tape drive, etc.

The CPU **20** includes logic for executing a plurality of program instructions and for executing a plurality of hardware and software interrupts. The hardware and software interrupts occur as the result of external events and system traps. Interrupts are serviced after execution of the current instruction in accordance with an interrupt service routine. The interrupts are classified into two types: maskable and non-maskable. Maskable interrupts are typically used to respond to asynchronous external hardware events and non-maskable interrupts are typically used to service very high priority events.

There is shown in the Fig. 2 a schematic memory map diagram of the RAM **34** when hardware **70** (the various components shown in the Fig. 1) of the computer system **10** is under the control of a generic or standard operating system. Typical standard operating systems include MS-DOS, Windows 3.x and Windows 95 (from Microsoft), DOS (various versions), OS/2 (from IBM Corporation), Unix (various versions) and NetWare (from Novell). In the more recent versions of most operating systems, at the most basic level is a kernel which is that portion of the

operating system which remains continuously in main memory and consists of the most frequently used part of the operating system. The kernel is responsible for the creation, deletion and state-switching of the many processes that define the computer's behavior by quickly responding to a steady flow of interrupt requests. The kernel is stored in a kernel portion **72** of the RAM **34**. At a higher level is a user application which is stored in a user applications portion **74** of the RAM **34**.

There is shown in the Fig. 3 the hardware **70** and a schematic memory map diagram of the RAM **34** when the computer system **10** is under the control of a Windows NT operating system (from Microsoft). At the most basic level is a hardware abstraction layer (HAL) portion **78**. Above the HAL **78** is the kernel portion **72** and the user applications portion **74**. Also stored in the RAM **34** is a timer portion **80** of the operating system which defines times at which various operating system events occur.

In order to execute the instructions of a program such as an operating system or a user application, the CPU **20** must first fetch the instruction from the memory **34**. The sequence of operations involved in processing an instruction constitutes an instruction cycle which is subdivided into a fetch cycle and an execution cycle. The instruction is obtained from main memory during the fetch cycle and the instruction cycle includes decoding the instruction, fetching any required operands and performing the operation specified by the instruction's opcode. The instruction cycle is defined by a sequence of microoperations with CPU cycle time being the time for the shortest well-defined such microoperation. The CPU cycle time is the basic unit for measuring CPU actions and the reciprocal of this time is the CPU clock rate. A typical CPU running at a 100 Mhz cycle rate will have a 10 nsec CPU cycle duration. As shown in the Fig. 4, a typical instruction cycle **82** includes in sequence a first CPU cycle "Fetch Instruction" **82a**, a second CPU cycle "Decode Instruction" **82b**, a third CPU cycle "Fetch Operand" **82c**, a fourth CPU cycle "Execute Instruction" **82d** and a fifth CPU cycle "Store Result" **82e**. At the end of any CPU cycle is a DMA (direct memory access) breakpoint **82f** or **82g** at which point the CPU **20** will yield control of the main bus

18 to a device in response to a DMA request received during that CPU cycle. The DMA breakpoint **82g** at the end of the fifth CPU cycle also is an interrupt breakpoint at which point the CPU **20** tests for the presence of an interrupt signal. If the interrupt signal test is positive, the CPU **20** stores the current status information and executes an interrupt service routine to process the interrupt signal received during the instruction cycle **82**. Of course, some instructions are longer and require an instruction cycle with more CPU cycles.

There is shown in the Fig. 5 a flow diagram of the operation of the operating systems shown in the memory maps of the Fig. 2 and the Fig. 3. The operating system begins at "START" **84** and enters a decision point "INST. ?" **86** wherein it is determined whether an instruction is to be executed. If no instruction is to be executed, the program branches at "NO" and re-enters the decision point **86** to again look for an instruction. If an instruction is to be executed, the program branches at "YES" to an instruction set "FETCH INSTRUCTION" **88** which corresponds to the CPU cycle **82a** shown in the Fig. 4. The program next moves to an instruction set "EXECUTE INSTRUCTION" **90** wherein the CPU cycles **82b** through **82e** of the instruction cycle **82** are performed.

The program then enters a decision point "INT ?" **92** to test for an interrupt. If no interrupt has occurred, the program branches from the decision point **92** at "NO" and enters an instruction set "STORE CURRENT STATUS INFORMATION" **94**. Now the program has entered the interrupt service portion of the operating system and the information currently being processed by the computer is stored as the current status information. The program enters an instruction set "EXECUTE INTERRUPT SERVICE ROUTINE" **96** wherein the interrupt is serviced in accordance with an associated set of stored interrupt service instructions executed during subsequent cycles. After the interrupt has been serviced, the program enters an instruction set "RECALL CURRENT STATUS INFORMATION" **98** wherein the previously stored program information is recalled into the computer memory. The program then returns to decision point **86** to look for the next program instruction to be executed.

The basic unit of computing managed by an operating system is a process or task which can be defined as a self-contained program module in the course of execution. The operating system must allocate multiple resources to and schedule the tasks. A clock, such as the real time clock **48** of the Fig. 1, generates an interrupt which can be used to define the duration of a time slice in which the operating system can execute instructions to perform a task. The use of time slices permits the CPU to multiplex tasks so that two or more tasks appear to be running at the same time. A typical hardware timer circuit defines a time slice duration of approximately 10 msec. Although the time slice duration can be reduced, computer system operating requirements usually dictate a lower limit of approximately one msec. As shown in the Fig. 6, a first standard time slice **100** extends from time zero to a time **t1**, a second standard time slice **100** extends from the time **t1** to a time **t2** and a third standard time slice **100** begins at the time **t2**. The time slices **100** and subsequent such time slices are of equal length or duration as defined by the timer. Each of the standard time slices **100** includes a processing portion **102** having a plurality of the instruction cycles **82** for executing program instructions. However, a part of the standard time slice **100** must be used to switch to the next time slice and the task to be performed therein which part is known as a CPU "overhead" portion "OH" **104**.

The duration of the "overhead" portion **104** remains relatively constant as the duration of the time slice **100** is decreased. There is shown in the Fig. 7 a plot **106** of time slice duration versus overhead. When the time slice duration has been reduced to the duration of the "overhead" portion **104**, the CPU overhead is 100%.

In process control systems, data exchange cycle times of less than one millisecond are known as "hard real time." As discussed above, the hardware timer used in a typical computer system is set to limit the time slice duration to approximately ten milliseconds, but could be set faster. The time slice duration is selected based upon the operating system performance and the CPU speed. If the clock rate is increased to a "hard" real time rate, the increase in CPU "overhead" would seriously impact the performance of the standard operating system and the

user programs. Also, commonly used operating systems do not run in a deterministic manner. That is, they do not run programs in a repeatable fashion which is required for real time control. Therefore, although the standard computer system **10** running a commercially available operating system has the computing power to run real time control programs, the interrupt response rate and non-deterministic operating system cannot provide the level of control required for many real time processes.

CNC equipment and manufacturing robots typically require data exchange on the order of every 250 microseconds. Servoactuators typically require data exchange on the order of every 100 microseconds. Therefore, the method and apparatus according to the present invention provides the precise control required by such manufacturing equipment while allowing the use of a standard operating system and standard computer hardware.

The method and apparatus according to the present invention solves the real time environment problems by reserving a portion of the standard microprocessor random access memory for running a real time software system program. The real time software system decreases the duration of the time slices to provide deterministic services and control the external data exchange at a "hard" real time interrupt rate permitting precise control of process equipment such as robot servomotors. The real time software system also provides time slices in which the regular operating system runs.

A real time control application must be able to service interrupts in "hard" real time and run "normal" control algorithms (i.e. non-interrupt driven) in a deterministic fashion. The standard operating system does not perform in this manner. The RTSS according to the present invention solves this problem by running a concurrent operating system with the same physical hardware as the standard operating system. The RTSS shares the CPU **20** with the standard operating system in a deterministic manner to provide deterministic services to the user. Thus, RTSS can provide libraries for implementing "hard" real time interrupt routines, libraries for high speed deterministic inter-task communication, libraries

for high speed deterministic inter-task synchronization, prioritized high speed deterministic task management, libraries for direct hardware access and libraries for communication with vendor specific hardware for process control.

There is shown in the Fig. 8 a schematic memory map diagram of a computer memory configuration including a real time software system in accordance with the present invention. The user applications portion **74** of the RAM **34** shown in the Fig. 3 has had two sections of memory divided therefrom. A first section or RTSS portion **76a** is available for an alternate operating system in accordance with the present invention and a second section or micro-kernel portion **76b** stores the kernel for the RTSS. The portions **76a** and **76b** can be smaller than the corresponding portions **74** and **72** for the standard operating system due to the relative size of the software programs. For example, if the RAM **34** has a 32 MB total capacity, the portions **72**, **74**, **78** and **80** can be a total of 30 MB and the portions **76a** and **76b** can be a reserved 2 MB portion of the total memory capacity.

There is shown in the Fig. 9 a timing diagram of the time slices for the configuration shown in the Fig. 8. A first RTSS time slice **108** is utilized to run the real time software system and a second OS time slice **110** is utilized to run the standard operating system. The RTSS takes over the system timer or the external non-maskable interrupt clock and utilizes the clock pulses which define the standard time slice **100** in the Fig. 6 to subdivide into shorter duration time slices by a high speed interrupt. For example, if the standard time slice **100** is one millisecond in duration, the time slices **108** and **110** can each be one half millisecond in duration. However, in order to provide "hard" real time control, the time slices **108** and **110** can be generated with durations in the microsecond range. For example, a time slice duration of 25 microseconds means that any real time interrupt to be serviced by the RTSS is delayed no more than the 25 microseconds duration of the OS time slice **110**. Such operation has the advantage of not changing the CPU "overhead" since the standard operating system still runs on the standard time slice **100**, but with one half of the processing time.

Furthermore, the time slices **108** and **110** do not have to be equal in number such that a duty cycle (the number the portions **108** divided by the sum of the number of the portions **108** and the number of the portions **110** in a predetermined period) can be varied to optimize the performance of the computer system **10**. Thus, the number of the RTSS time slices **108** can be increased to accommodate more processing time as long as the degradation in the response time of the program being executed during the standard operating system time slices **110** can be tolerated. A method of changing the duty cycle is shown in the Fig. 10 wherein two RTSS time slices **108** are generated for each OS time slice **110**. This ratio can be selected to provide the desired performance.

The duration of the time slices **108** and **110** can be determined by an optional external timer circuit that utilizes a non-maskable interrupt to generate a timing signal for the real time software system task switcher to use to divide the standard time slice **100**. For example, as shown in the Fig. 1, an RTSS timer **112** can be connected to the main bus **18** to respond to the interrupt generated by the real time clock **48** for determining the duration of the standard time slice **100.** The timer **112** is setable or programmable for generating non-maskable interrupts at a rate which defines the duration of the time slices **108** and **110**. Of course, the timer **112** can be connected to the computer system **10** in any suitable manner in order to generate the non-maskable interrupts.

The method of operating the computer system **10** according to the present invention, the computer system having the memory **34** for storing an operating system and the CPU **20** for executing program instructions under control of the operating system, comprising the steps of: a. storing a first operating system in the first portion **72** of the memory; b. operating the CPU under the control of the first operating system to generate a plurality of standard time slices **100**; the method characterized by the steps of: c. storing a second operating system in the second portion **76a,76b** of the memory; d. operating the CPU under the control of the second operating system concurrently with the first operating system to generate at least one first time slice **110** and at least one second time slice **108** during each of

the standard time slices; e. executing at least one program instruction in accordance with the first operating system during the first time slice; and f. executing at least one other program instruction in accordance with the second operating system during the second time slice.

The computer system **10** according to the present invention includes the CPU **20** connected to the memory **34** for executing program instructions according to an operating system comprising: the memory having the first portion **72** in which the first operating system is stored and the second portion **76a,76b** in which the second operating system is stored; and the CPU operating under control of the first operating system to sequentially generate the standard time slices **100** and operating concurrently under control of the second operating system to divide the standard time slices into at least one of the first time slices **110** and at least one of the second time slices **108**, the CPU executing first program instructions during the first time slice and executing second program instructions during the second time slice.

In accordance with the provisions of the patent statutes, the present invention has been described in what is considered to represent its preferred embodiment. However, it should be noted that the invention can be practiced otherwise than as specifically illustrated and described without departing from its spirit or scope.

WHAT IS CLAIMED IS:

1. A method of operating a computer system **(10)**, the computer system having a memory **(34)** for storing an operating system and a CPU **(20)** for executing program instructions under control of the operating system, comprising the steps of:

a. storing a first operating system in a first portion **(72)** of the memory **(34)**;

b. operating the CPU **(20)** under the control of the first operating system to generate a plurality of standard time slices **(100)**;

the method characterized by the steps of:

c. storing a second operating system in a second portion **(76a,76b)** of the memory **(34)**;

d. operating the CPU **(20)** under the control of the second operating system concurrently with the first operating system to generate at least one first time slice **(110)** and at least one second time slice **(108)** during each of the standard time slices **(100)**;

e. executing at least one program instruction in accordance with the first operating system during the first time slice **(110)**; and

f. executing at least one other program instruction in accordance with the second operating system during the second time slice **(108)**.

2. The method according to claim 1 wherein the step b. is performed by setting a duration of the standard time slices **(100)** equal to approximately one millisecond.

3. The method according to claim 1 wherein the step d. is performed by setting a duration of the first time slice **(110)** and a duration of the second time slice **(108)** to approximately twenty-five microseconds each.

4. The method according to claim 1 wherein the step d. is performed by subdividing the standard time slice **(100)** equally among a predetermined number of the first time slices **(110)** alternating with the predetermined number of the second time slices **(108)**.

5. The method according to claim 1 wherein the step d. is performed by allocating the standard time slice **(100)** between a first predetermined number of the first time slices **(110)** and a second predetermined number of the second time slices **(108)**.

6. The method according to claim 5 including a step of selecting a duty cycle value and wherein the step d. is performed by determining the first predetermined number of the first time slices **(110)** and the second predetermined number of the second time slices **(110)** according to the duty cycle value.

7. The method according to claim 1 wherein the second operating system executes program instructions during the second time slice **(108)** permitting performance of tasks in real time.

8. The method according to claim 1 wherein the step f. includes servicing a plurality of input/output interrupts in response to the program instructions executed during the second time slices **(108)**.

9. The method according to claim 1 wherein the first operating system is a standard operating system for operating the CPU **(20)** to sequentially generate the standard time slices **(100)** of a predetermined duration each including a plurality of instruction cycles **(82)**.

10. The method according to claim 9 wherein the second operating system is a real time software system for operating the CPU **(20)** to divide the

standard time slices **(100)** into at least one real time software system time slice **(108)** and at least one standard operating system time slice **(110)**, each of the real time software system time slice and the standard operating system time slice including at least one of the instruction cycles **(82)**.

11. The method according to claim 1 wherein the step d. is performed by generating a non-maskable interrupt signal at a rate setting a duration of the first time slices **(110)** and the second time slices **(108)** in response to an interrupt signal setting a duration of the standard time slices **(100)**.

12. The method according to claim 1 wherein the second operating system operates the CPU **(20)** to provide deterministic services to a user of the computer system **(10)**.

13. A computer system **(10)** including a CPU **(20)** connected to a memory **(34)** for executing program instructions according to an operating system comprising:

the memory **(34)** having a first portion **(72)** in which a first operating system is stored and a second portion **(76a,76b)** in which a second operating system is stored; and

the CPU **(20)** operating under control of said first operating system to sequentially generate standard time slices **(100)** and operating concurrently under control of said second operating system to divide said standard time slices into at least one first time slice **(110)** and at least one second time slice **(108)**, the CPU executing at least one first program instruction during said first time slice and executing at least one second program instruction during said second time slice.

14. The apparatus according to claim 13 wherein said first operating system is a standard operating system and said second operating system is a real time software system.

15. The apparatus according to claim 13 wherein said first operating system sets a duration of said standard time slices **(100)** equal to approximately one millisecond.

16. The apparatus according to claim 13 wherein said second operating system sets a duration of said first time slice **(110)** and a duration of said second time slice **(108)** to approximately twenty-five microseconds each.

17. The apparatus according to claim 13 wherein said second operating system controls the CPU **(20)** to subdivide said standard time slice **(100)** equally among a predetermined number of said first time slices **(110)** alternating with said predetermined number of said second time slices **(108)**.

18. The apparatus according to claim 13 wherein said second operating system selects a duty cycle value and controls the CPU **(20)** to subdivide said standard time slice **(100)** among a plurality of said first time slices **(110)** and said second time slices **(108)** in proportion according to said duty cycle value.

19. The apparatus according to claim 13 wherein the CPU **(20)** is connected to a plurality of devices for generating input/output interrupts and said second operating system controls the CPU to service said input/output interrupts in response to said second program instruction executed during said second time slice **(108)**.

20. The apparatus according to claim 13 including a timer **(112)** connected to the CPU **(20)** for generating a non-maskable interrupt signal at a rate setting a duration of said first time slice **(110)** and said second time slice **(108)** in response to an interrupt signal setting a duration of said standard time slices **(100)**.

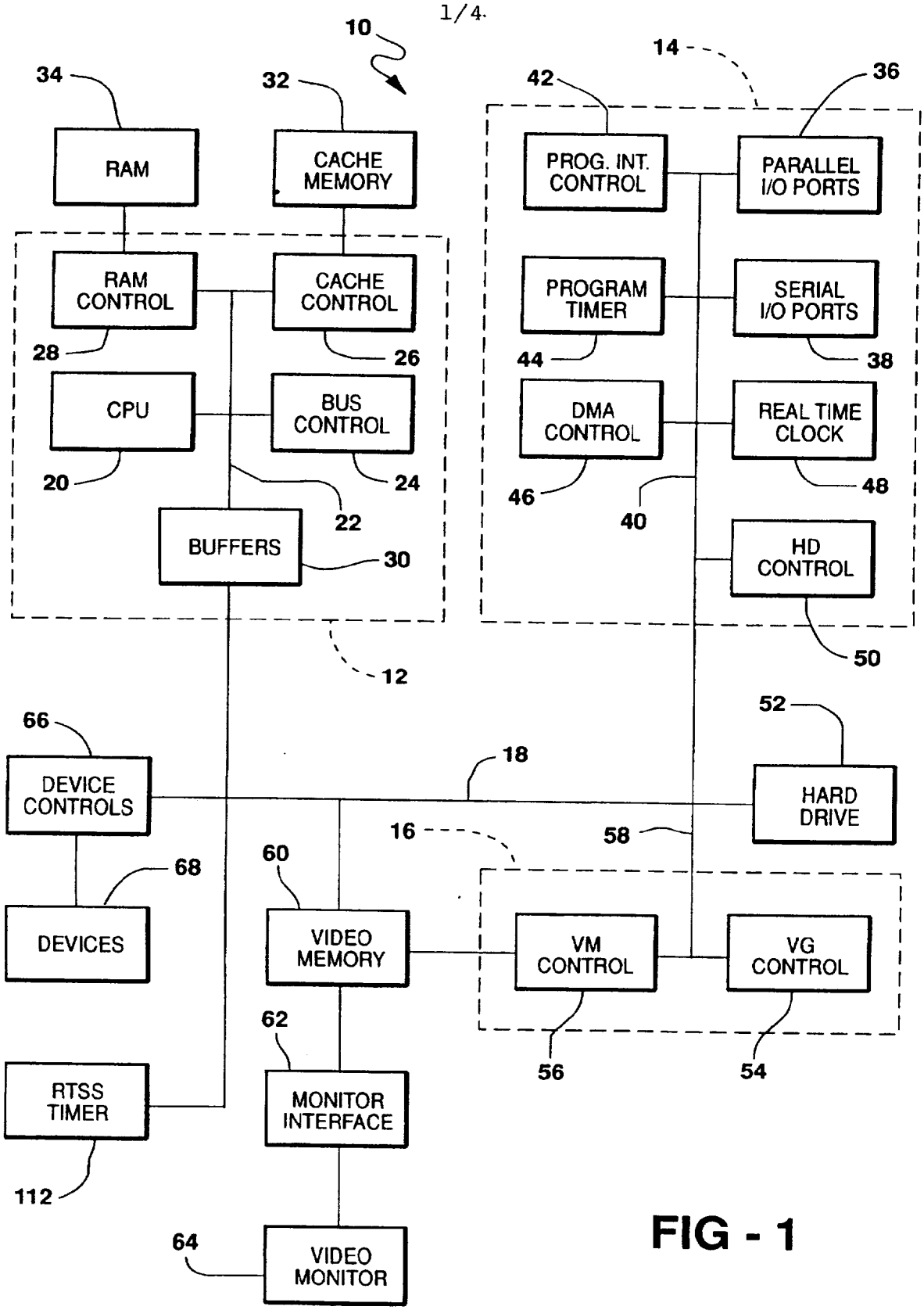

10
34
32
14
42
36
RAM
CACHE MEMORY
PROG. INT. CONTROL
PARALLEL I/O PORTS
RAM CONTROL
CACHE CONTROL
PROGRAM TIMER
SERIAL I/O PORTS
28
26
44
38
CPU
BUS CONTROL
DMA CONTROL
REAL TIME CLOCK
20
24
46
48
40
22
BUFFERS
30
HD CONTROL
50
12
66
52
18
DEVICE CONTROLS
HARD DRIVE
16
58
60
68
DEVICES
VIDEO MEMORY
VM CONTROL
VG CONTROL
62
56
54
RTSS TIMER
MONITOR INTERFACE
112
64
VIDEO MONITOR

FIG - 1

FIG - 2
PRIOR ART

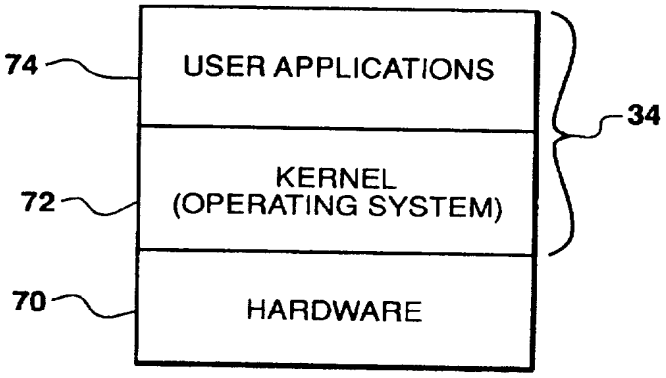


FIG - 3
PRIOR ART

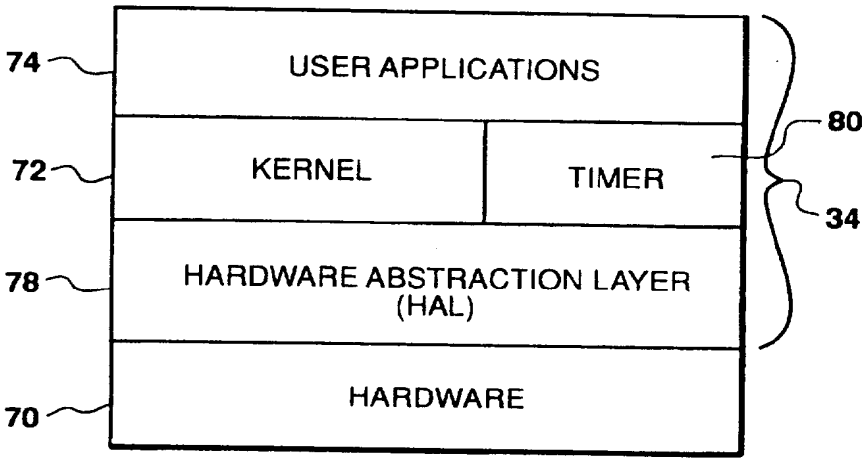


FIG - 4

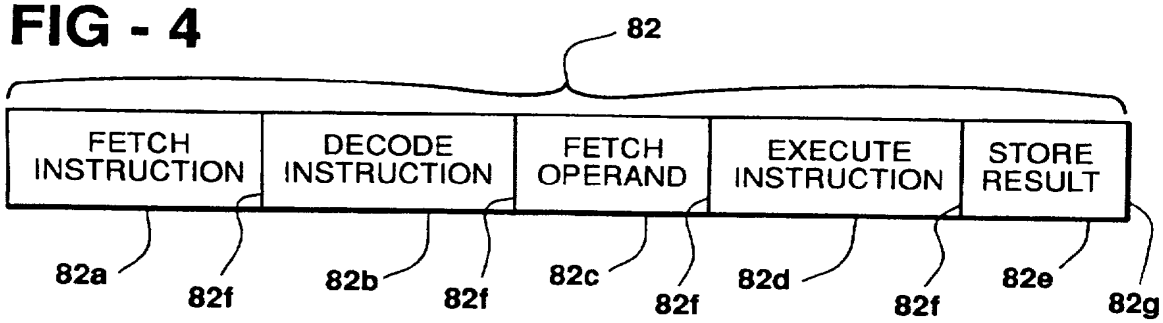


FIG - 6

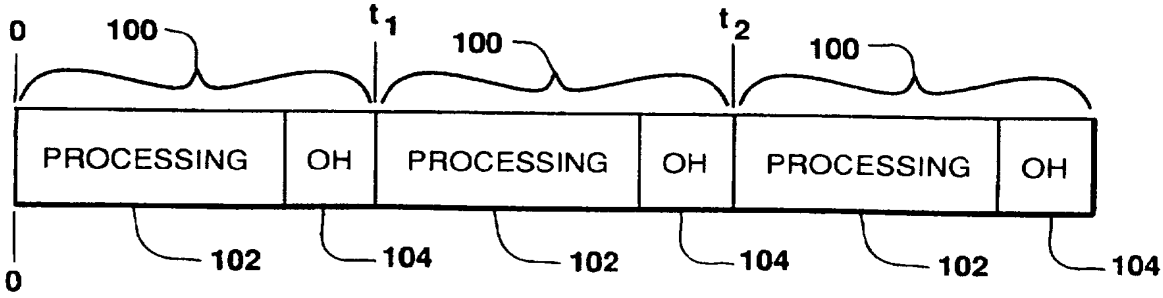


FIG - 5

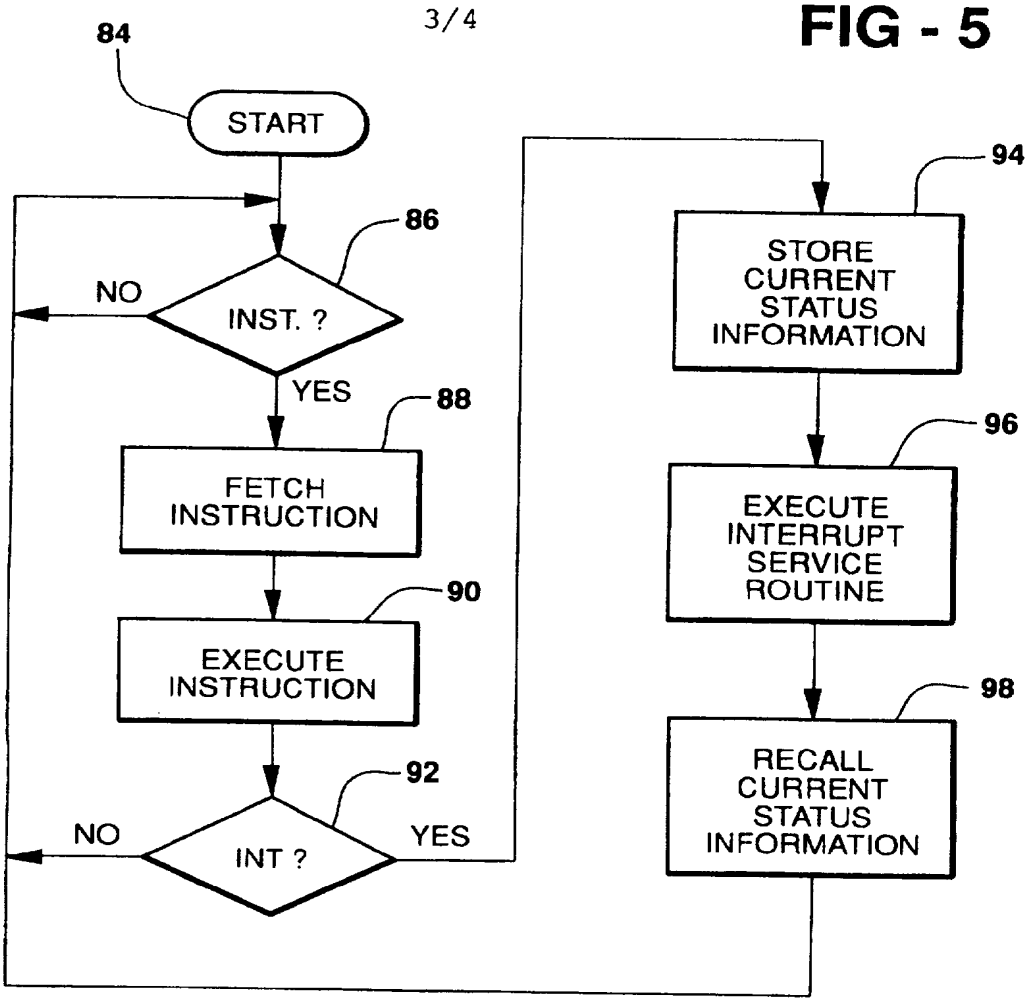
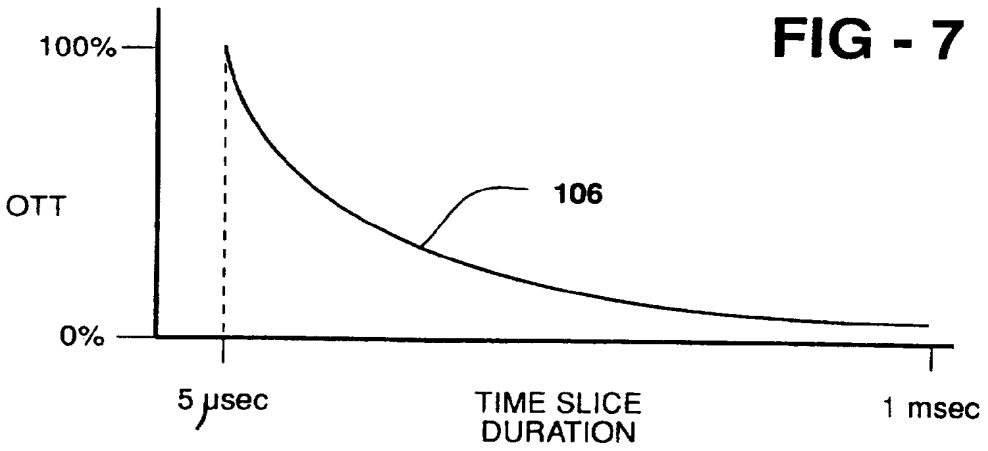

84
START
86
INST. ?
NO
YES
88
FETCH INSTRUCTION
90
EXECUTE INSTRUCTION
92
INT ?
NO
YES
94
STORE CURRENT STATUS INFORMATION
96
EXECUTE INTERRUPT SERVICE ROUTINE
98
RECALL CURRENT STATUS INFORMATION

FIG - 7


100%
OTT
0%
106
5 µsec
TIME SLICE DURATION
1 msec

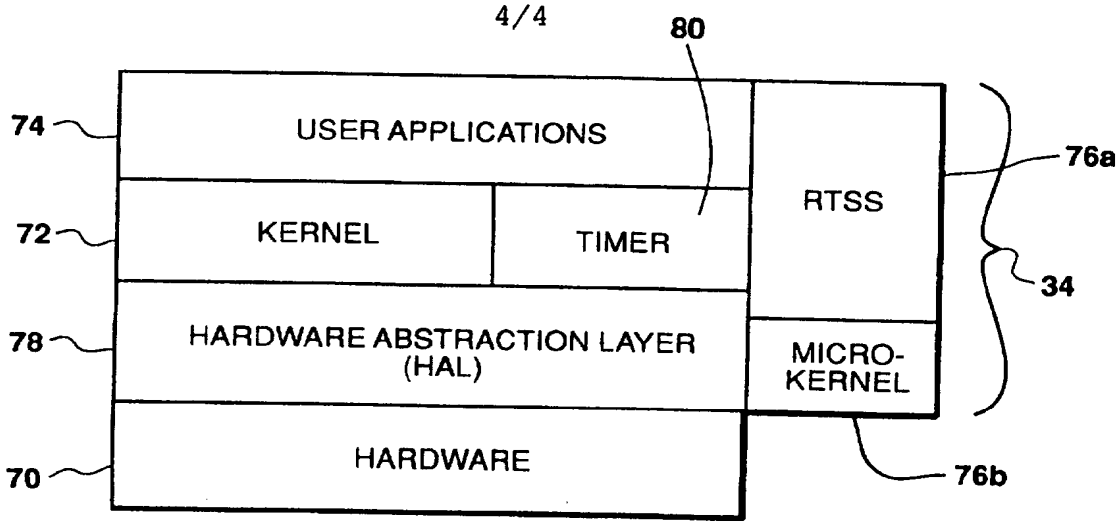


FIG - 8

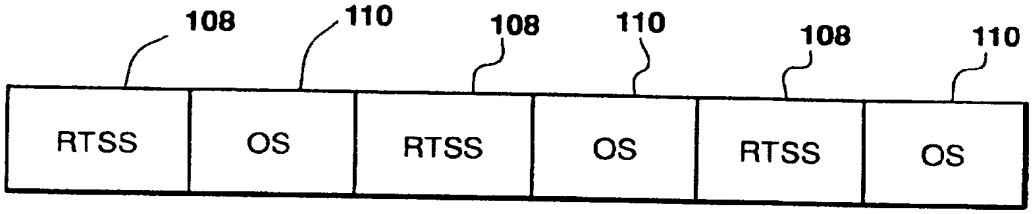


FIG - 9

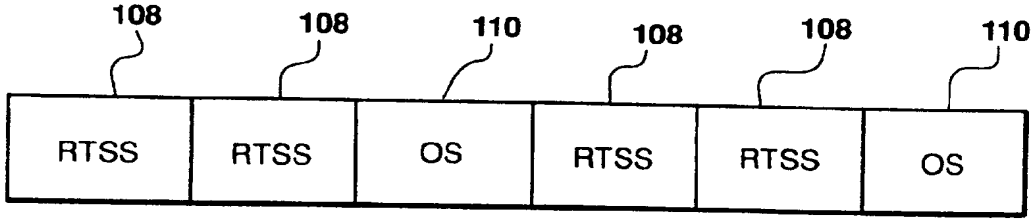


FIG - 10

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/13182

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) : G06F 15/16
US CL : 395/677, 673
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 395/677, 674, 673, 672, 670

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
APS, IEEE PUBLICATIONS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,528,513 A (VAITZBLIT ET AL) 18 June 1996, entire document, especially column 6, line 4 to column 7, line 33, figures 3-4.	1-20
Y	US 5,394,547 A (CORRENTI ET AL) 28 February 1995, see entire document, especially column 2, lines 25-46.	1-20
Y	US 5,222,215 A (CHOU ET AL) 22 June 1993, see Abstract, figure 1, column 3, lines 7-17.	1-20
Y	SHAH, K. Windows DDE for real-time applications, Dr. Dobb's Journal, January 1993, Vol 18, No. 1, p58(4), see first paragraph under section "iRMX for Windows DDE Architecture".	1-20

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search
06 NOVEMBER 1997

Date of mailing of the international search report
12 JAN 1998

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
SUE X. LAO
Telephone No. (703) 305 9657

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/13182

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	POTTER, D., Designing a real-time debugger, the best of both worlds. Dr. Dobb's Journal, November 1992, Vol. 17, No. 11 p70(7), see first paragraph, and the fourth and fifth paragraphs under section "Partitioning the Debugger".	12
Y	SILBERSCHATZ, A, et al, Operating System Concepts, Addison-Wesley Publishing Company, 1994, see pages 143-145.	9-11, 13-14, 16, 18-20
Y	Author admitted prior art, pages 1-3.	2, 15
Y	US 5,437,039 A (YUEN) 25 July 1995, see abstract.	4, 17
A	US 4,736,318 A (DELYANI ET AL) 05 April 1988.	1-20
A	US 4,908,750 A (JABLOW) 13 March 1990.	1-20
A	SPRUNT, B., et al, Exploring Unused Periodic Time For Aperiodic Services Using The Extended Priority Exchange Algorithm, IEEE Ondisc, 1988, pp 251-258.	1-20